7. dplyr에 의한 자료 다듬기

통계 데이터 세트의 구조는 변수가 열, 관찰값이 행을 이루고 있는 2차원 구조이며, 따라서 데이터 프레임으로 입력된다. 일반적으로 R에 입력된 데이터 세트는 바로 통계분석이 가능한 상태가 아닌 경우가 대부분이다. 분석에 필요한 적절한 변수가 없거나, 특정 조건을 만족하는 관찰값만을 선택해야 하거나, 관찰값의 순서를 바꿔야 하거나, 그룹별로 자료를 합쳐야 하는 등등의 작업이 필요한 경우가 대부분이다. 더욱이 최적의 분석을 위한 적절한 자료 변형 혹은 데이터 다듬기는 시간이 매우 많이 소요되는 힘든 작업이다. 따라서 일관된 법칙에 따라 편리하게 적용할 수 있는 데이터 다듬기 기법이 절실하게 필요한 상황이라고 하겠다.

이번 장에서는 패키지 dplyr을 이용하여 데이터 프레임을 대상으로 하는 다양한 데이터 다듬기 기법을 살펴보고자 한다. 패키지 dplyr은 core tidyverse에 속한 패키지이므로 사용하기 위해서는 library(tidyverse)를 실행하면 된다. 데이터 다듬기에 사용되는 함수에는 조건에 따라 관찰값을 선택하는 filter(), 관찰값을 정렬하는 arrange(), 변수를 선택하는 select(), 새로운 변수를 추가하는 mutate(), 그리고 그룹의 생성하는 group_by()와 자료를 요약하는 summarize()가 있다.

7.1 조건에 따른 관찰값의 선택: filter()

함수 filter()를 이용하면, 특정 조건을 만족시키는 관찰값을 선택할 수 있다. 함수의 첫 번째 입력 요소는 데이터 프레임이고, 두 번째 요소는 관찰값을 선택하는 조건이 입력된다. 특정 조건을 설정할 때는 비교 연산자(>, >=, <, <=, !=, ==)와 논리 연산자(&, |, !)가 사용되며, 또한 연산자 %in%가 매우 유용하게 사용된다. 예제를 통해 자세한 사용법을 확인해 보자.

- 예제: mtcars

변수 mpg의 값이 30 이상인 자동차를 선택해 보자. 함수 filter()에 전통적인 데이터 프레임을 입력하면 결과가 데이터 프레임이 되고, tibble을 입력하면 결과가 tibble로 출력된다. 데이터 프레임 mtcars를 tibble로 전환시키고 선택 조건과 더불어 함수 filter()에 입력해 보자.

```
> mtcars_t <- as_tibble(mtcars)

> filter(mtcars_t, mpg >= 30)
# A tibble: 4 x 11
    mpg   cyl   disp    hp   drat    wt   qsec    vs    am  gear  carb
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1  32.4  4.00  78.7   66.0  4.08  2.20  19.5  1.00  1.00  4.00  1.00
2  30.4  4.00  75.7   52.0  4.93  1.62  18.5  1.00  1.00  4.00  2.00
3  33.9  4.00  71.1   65.0  4.22  1.84  19.9  1.00  1.00  4.00  1.00
4  30.4  4.00  95.1  113    3.77  1.51  16.9  1.00  1.00  5.00  2.00
```

변수 mpg의 값이 30 이상이고 변수 wt의 값이 1.8 미만인 자동차를 선택해 보자.

```
> filter(mtcars_t, mpg >= 30 & wt < 1.8)
# A tibble: 2 x 11
    mpg   cyl   disp    hp   drat    wt    qsec    vs    am   gear  carb
  <db1> <db1> <db1> <db1> <db1> <db1> <db1> <db1> <db1> <db1> <db1>
1  30.4  4.00  75.7  52.0   4.93   1.62  18.5   1.00  1.00  4.00  2.00
2  30.4  4.00  95.1 113     3.77   1.51  16.9   1.00  1.00  5.00  2.00
```

변수 mpg의 값이 30 이하이고, 변수 cyl의 값이 6 또는 8이며, 변수 am의 값이 1인 자동차를 선택해 보자. 논리 연산자 '&' 대신에 콤마를 사용할 수 있으며, 'cyl == 6|cyl == 8' 보다는 'cyl %in% c(6,8)'이 훨씬 간단하게 조건을 설정하는 방법이 된다.

```
> filter(mtcars_t, mpg <= 30, cyl %in% c(6,8), am == 1)
# A tibble: 5 x 11
    mpg    cyl  disp    hp   drat    wt    qsec    vs    am   gear  carb
  <db1> <db1> <db1> <db1> <db1> <db1> <db1> <db1> <db1> <db1> <db1>
1  21.0  6.00   160   110   3.90   2.62   16.5     0   1.00  4.00  4.00
2  21.0  6.00   160   110   3.90   2.88   17.0     0   1.00  4.00  4.00
3  15.8  8.00   351   264   4.22   3.17   14.5     0   1.00  5.00  4.00
4  19.7  6.00   145   175   3.62   2.77   15.5     0   1.00  5.00  6.00
5  15.0  8.00   301   335   3.54   3.57   14.6     0   1.00  5.00  8.00
```

변수 mpg의 값이 mpg의 중앙값과 Q_3 사이에 있는 자동차를 선택해 보자. 여기서 Q_3는 0.75분위수를 의미하는 것으로, p 분위수란 전체 데이터 중 $100 \times \mathrm{p}\%$ 데이터 보다는 크고, 나머지 $100 \times (1-\mathrm{p})\%$ 데이터 보다는 작은 수를 의미한다. 분위수의 계산은 함수 quantile()로 하는데, 벡터 x의 0.75분위수의 계산은 quantile(x, probs=0.75)로 할 수 있다.

```
> filter(mtcars_t, mpg >= median(mpg) & mpg <= quantile(mpg,probs=0.75))
# A tibble: 10 x 11
     mpg   cyl  disp    hp   drat    wt   qsec    vs    am   gear  carb
   <db1> <db1> <db1> <db1> <db1> <db1> <db1> <db1> <db1> <db1> <db1>
 1  21.0  6.00   160   110   3.90  2.62  16.5  0      1.00  4.00  4.00
 2  21.0  6.00   160   110   3.90  2.88  17.0  0      1.00  4.00  4.00
 3  22.8  4.00   108    93.0 3.85  2.32  18.6  1.00   1.00  4.00  1.00
 4  21.4  6.00   258   110   3.08  3.22  19.4  1.00   0     3.00  1.00
 5  22.8  4.00   141    95.0 3.92  3.15  22.9  1.00   0     4.00  2.00
 6  19.2  6.00   168   123   3.92  3.44  18.3  1.00   0     4.00  4.00
 7  21.5  4.00   120    97.0 3.70  2.46  20.0  1.00   0     3.00  1.00
 8  19.2  8.00   400   175   3.08  3.84  17.0  0      0     3.00  2.00
 9  19.7  6.00   145   175   3.62  2.77  15.5  0      1.00  5.00  6.00
10  21.4  4.00   121   109   4.11  2.78  18.6  1.00   1.00  4.00  2.00
```

벡터 x가 특정 두 숫자 사이에 있는지를 확인하는 방법은 'x >= left & x <= right'가 되는데, 이것을 함수 between()을 이용해서 between(x, left, right)로 할 수 있다. 따라서 위 예제는 다음과 같이 실행해도 동일한 결과를 얻는다.

```
> filter(mtcars_t, between(mpg, median(mpg), quantile(mpg,probs=0.75)))
```

- 예제: airquality

데이터 프레임 airquality를 airs라는 이름의 tibble로 전환하고, 처음 5개 케이스를 출력해 보자.

```
> airs <- as_tibble(airquality)

> print(airs,n=5)
# A tibble: 153 x 6
  Ozone Solar.R  Wind  Temp Month   Day
  <int>   <int> <dbl> <int> <int> <int>
1    41     190  7.40    67     5     1
2    36     118  8.00    72     5     2
3    12     149 12.6     74     5     3
4    18     313 11.5     62     5     4
5    NA      NA 14.3     56     5     5
# ... with 148 more rows
```

변수 Ozone 또는 Solar.R이 결측값인 관찰값을 선택해 보자. 결측값의 확인은 함수 is.na()로 할 수 있다.

```
> filter(airs, is.na(Ozone) | is.na(Solar.R))
# A tibble: 42 x 6
   Ozone Solar.R  Wind  Temp Month   Day
   <int>   <int> <dbl> <int> <int> <int>
 1    NA      NA 14.3     56     5     5
 2    28      NA 14.9     66     5     6
 3    NA     194  8.60    69     5    10
 4     7      NA  6.90    74     5    11
 5    NA      66 16.6     57     5    25
 6    NA     266 14.9     58     5    26
 7    NA      NA  8.00    57     5    27
 8    NA     286  8.60    78     6     1
 9    NA     287  9.70    74     6     2
10    NA     242 16.1     67     6     3
# ... with 32 more rows
```

7.2 관찰값의 정렬: arrange()

함수 arrange()는 특정 변수를 기준으로 데이터 프레임의 행을 재배열할 때 사용된다. 함수의 첫 번째 입력 요소는 데이터 프레임이고 두 번째 요소는 정렬의 기준이 되는 변수가 된다. 2개 이상의 정렬 기준 변수를 입력하게 되면, 추가된 변수는 앞선 변수가 같은 값을 갖는 관찰값들의 정렬 기준으로 사용된다. 정렬은 오름차순이 디폴트이며, 내림차순으로 배열하고자 할 때에는 기준 변수를 함수 desc()에 입력해야 한다.

- 예제: mtcars

데이터 프레임 mtcars를 tibble로 전환하고 변수 mpg의 값이 가장 좋지 않은 자동차부터 다시 배열해 보자. 연비가 좋지 않다는 것은 연비가 낮다는 것을 의미하는 것이므로 변수 mpg를 오름차순 정렬 변수로 사용하면 된다.

```
> mtcars_t <- as_tibble(mtcars)

> arrange(mtcars_t, mpg)
# A tibble: 32 x 11
     mpg   cyl   disp    hp   drat    wt   qsec    vs    am    gear  carb
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  10.4  8.00   472   205   2.93  5.25   18.0     0     0      3.00  4.00
 2  10.4  8.00   460   215   3.00  5.42   17.8     0     0      3.00  4.00
 3  13.3  8.00   350   245   3.73  3.84   15.4     0     0      3.00  4.00
 4  14.3  8.00   360   245   3.21  3.57   15.8     0     0      3.00  4.00
 5  14.7  8.00   440   230   3.23  5.34   17.4     0     0      3.00  4.00
 6  15.0  8.00   301   335   3.54  3.57   14.6     0     1.00   5.00  8.00
 7  15.2  8.00   276   180   3.07  3.78   18.0     0     0      3.00  3.00
 8  15.2  8.00   304   150   3.15  3.44   17.3     0     0      3.00  2.00
 9  15.5  8.00   318   150   2.76  3.52   16.9     0     0      3.00  2.00
10  15.8  8.00   351   264   4.22  3.17   14.5     0     1.00   5.00  4.00
# ... with 22 more rows
```

데이터 프레임 mtcars_t를 변수 mpg의 값이 가장 좋지 않은 자동차부터 다시 배열을 하되, mpg 값이 같은 자동차의 경우에는 변수 wt의 값이 높은 자동차부터 배열해 보자. 변수 wt를 두 번째 정렬 기준 변수로 입력하되, 내림차순 정렬이 필요하므로 함수 desc()를 이용한다.

```
> arrange(mtcars_t, mpg, desc(wt))
# A tibble: 32 x 11
     mpg   cyl   disp    hp   drat    wt   qsec    vs    am    gear  carb
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  10.4  8.00   460   215   3.00  5.42   17.8     0     0      3.00  4.00
 2  10.4  8.00   472   205   2.93  5.25   18.0     0     0      3.00  4.00
 3  13.3  8.00   350   245   3.73  3.84   15.4     0     0      3.00  4.00
 4  14.3  8.00   360   245   3.21  3.57   15.8     0     0      3.00  4.00
 5  14.7  8.00   440   230   3.23  5.34   17.4     0     0      3.00  4.00
 6  15.0  8.00   301   335   3.54  3.57   14.6     0     1.00   5.00  8.00
 7  15.2  8.00   276   180   3.07  3.78   18.0     0     0      3.00  3.00
 8  15.2  8.00   304   150   3.15  3.44   17.3     0     0      3.00  2.00
 9  15.5  8.00   318   150   2.76  3.52   16.9     0     0      3.00  2.00
10  15.8  8.00   351   264   4.22  3.17   14.5     0     1.00   5.00  4.00
# ... with 22 more rows
```

- 예제: airquality

데이터 프레임 airquality를 tibble로 전환하고, 5월 1일부터 5월 10일까지의 자료만을 대상으로 변수 Ozone의 값이 가장 낮았던 날부터 다시 배열해 보자.

```
> airs <- as_tibble(airquality)
```

```
> airs_1 <- filter(airs, Month==5, Day<=10)

> arrange(airs_1, Ozone)
# A tibble: 10 x 6
   Ozone Solar.R  Wind  Temp Month   Day
   <int>   <int> <dbl> <int> <int> <int>
 1     8      19  20.1    61     5     9
 2    12     149  12.6    74     5     3
 3    18     313  11.5    62     5     4
 4    19      99  13.8    59     5     8
 5    23     299   8.60   65     5     7
 6    28      NA  14.9    66     5     6
 7    36     118   8.00   72     5     2
 8    41     190   7.40   67     5     1
 9    NA      NA  14.3    56     5     5
10    NA     194   8.60   69     5    10
```

위의 정렬 결과를 보면 변수 Ozone이 결측값인 케이스가 가장 뒤로 배열되어 있다. 이것이 문제는 아니지만, 어떤 경우에는 결측값인 케이스를 가장 앞으로 배열하는 것이 필요할 때도 있다. 데이터 프레임 airs_1을 변수 Ozone이 결측값이 있는 케이스부터 다시 배열해 보자. 이것은 배열 기준으로 논리형 벡터를 사용해야 되는 문제인데, TRUE와 FALSE의 배열에서 FALSE가 우선 순위에 있기 때문에 !is.na(Ozone)을 배열 기준으로 사용하면 된다.

```
> arrange(airs_1, !is.na(Ozone))
# A tibble: 10 x 6
   Ozone Solar.R  Wind  Temp Month   Day
   <int>   <int> <dbl> <int> <int> <int>
 1    NA      NA  14.3    56     5     5
 2    NA     194   8.60   69     5    10
 3    41     190   7.40   67     5     1
 4    36     118   8.00   72     5     2
 5    12     149  12.6    74     5     3
 6    18     313  11.5    62     5     4
 7    28      NA  14.9    66     5     6
 8    23     299   8.60   65     5     7
 9    19      99  13.8    59     5     8
10     8      19  20.1    61     5     9
```

이번에는 데이터 프레임 airs_1을 변수 Ozone의 값이 가장 높은 날부터 다시 배열하되 결측값이 있는 케이스를 가장 앞으로 배열해 보자. 이것은 !is.na(Ozone)을 첫 번째 배열 기준으로 하여 결측값이 있는 케이스를 가장 앞으로 배치하고, desc(Ozone)을 두 번째 배열 기준으로 입력해서 결측값이 아닌 케이스들을 변수 Ozone의 내림차순으로 다시 배열해야 되는 문제이다.

```
> arrange(airs_1, !is.na(Ozone), desc(Ozone))
# A tibble: 10 x 6
   Ozone Solar.R  Wind  Temp Month   Day
   <int>   <int> <dbl> <int> <int> <int>
 1    NA      NA  14.3    56     5     5
```

```
 2    NA     194    8.60    69     5    10
 3    41     190    7.40    67     5     1
 4    36     118    8.00    72     5     2
 5    28      NA   14.9     66     5     6
 6    23     299    8.60    65     5     7
 7    19      99   13.8     59     5     8
 8    18     313   11.5     62     5     4
 9    12     149   12.6     74     5     3
10     8      19   20.1     61     5     9
```

7.3 변수의 선택: select()

데이터 세트의 크기가 커짐에 따라 변수의 개수가 수백 또는 수천이 되는 경우를 접하는 것은 더 이상 드문 상황은 아니다. 이러한 경우, 분석에 필요한 변수를 선택하여 데이터 세트의 크기를 줄이는 것이 가장 먼저 해야 할 작업이 될 것이다. 함수 select()를 이용하면 변수 이름의 문자열 매칭을 통하여 매우 효과적으로 변수 선택을 할 수 있다. 함수의 첫 번째 입력 요소는 데이터 프레임이고 선택할 변수의 이름 또는 문자열 매칭과 관련된 함수가 그 다음에 나열된다.

- 예제: mtcars

데이터 프레임 mtcars의 row names를 변수 rowname으로 전환하고, 변수 rowname과 mpg만을 선택해 보자.

```
> mtcars_t <- as_tibble(mtcars)

> mtcars_t <- rownames_to_column(mtcars_t,var="rowname")

> select(mtcars_t, rowname, mpg)
# A tibble: 32 x 2
   rowname             mpg
   <chr>             <dbl>
 1 Mazda RX4          21.0
 2 Mazda RX4 Wag      21.0
 3 Datsun 710         22.8
 4 Hornet 4 Drive     21.4
 5 Hornet Sportabout  18.7
 6 Valiant            18.1
 7 Duster 360         14.3
 8 Merc 240D          24.4
 9 Merc 230           22.8
10 Merc 280           19.2
# ... with 22 more rows
```

데이터 프레임 mtcars_t의 첫 번째 변수인 mpg부터 여섯 번째 변수인 wt까지 모두 선택을 한다면 콜론 연산자를 사용할 수 있다.

```
> select(mtcars_t, mpg:wt)
# A tibble: 32 x 6
     mpg   cyl   disp    hp   drat    wt
   <db1> <db1> <db1> <db1> <db1> <db1>
 1  21.0  6.00   160   110    3.90  2.62
 2  21.0  6.00   160   110    3.90  2.88
 3  22.8  4.00   108    93.0  3.85  2.32
 4  21.4  6.00   258   110    3.08  3.22
 5  18.7  8.00   360   175    3.15  3.44
 6  18.1  6.00   225   105    2.76  3.46
 7  14.3  8.00   360   245    3.21  3.57
 8  24.4  4.00   147    62.0  3.69  3.19
 9  22.8  4.00   141    95.0  3.92  3.15
10  19.2  6.00   168   123    3.92  3.44
# ... with 22 more rows
```

데이터 세트에서 제거하고자 하는 변수는 이름에 마이너스 기호를 붙이면 된다. 변수 rowname과 qsec에서 carb까지를 제거해 보자.

```
> select(mtcars_t, -rowname, -(qsec:carb))
# A tibble: 32 x 6
     mpg   cyl  disp    hp   drat    wt
   <db1> <db1> <db1> <db1> <db1> <db1>
 1  21.0  6.00   160   110    3.90  2.62
 2  21.0  6.00   160   110    3.90  2.88
 3  22.8  4.00   108    93.0  3.85  2.32
 4  21.4  6.00   258   110    3.08  3.22
 5  18.7  8.00   360   175    3.15  3.44
 6  18.1  6.00   225   105    2.76  3.46
 7  14.3  8.00   360   245    3.21  3.57
 8  24.4  4.00   147    62.0  3.69  3.19
 9  22.8  4.00   141    95.0  3.92  3.15
10  19.2  6.00   168   123    3.92  3.44
# ... with 22 more rows
```

데이터 세트에 변수의 수가 대단히 많은 경우, 선택하고자 하는 변수 이름을 일일이 나열해서 선택하는 것은 상당히 비효율적인 작업 방식이라고 하겠다. 따라서 변수 이름을 효율적으로 선택할 수 있는 방법이 필요한데, 다음의 함수를 이용하면 문자열 매칭 작업을 편리하게 할 수 있다.

- starts_with("x"): 이름이 "x"로 시작하는 변수 선택
- ends_with("x"): 이름이 "x"로 끝나는 변수 선택
- contains("x"): 이름에 "x"가 있는 변수 선택
- num_range("x", 1:10): x1, x2, ..., x10과 동일

따라서 위에서 소개된 함수를 이용하여 데이터 프레임 mtcars_t에서

- “m”으로 이름이 시작하는 변수 선택과
- “p”로 이름이 끝나는 변수 선택과
- “a”가 이름에 있는 변수 선택 작업은 각각 다음과 같이 간편하게 할 수 있다.

```
> select(mtcars_t, starts_with("m"))
> select(mtcars_t, ends_with("p"))
> select(mtcars_t, contains("a"))
```

- 예제: iris

데이터 프레임 iris는 3종류의 붓꽃 각 50송이씩 150송이 붓꽃의 꽃받침 길이와 폭, 꽃잎 길이와 폭을 측정한 데이터이다. 변수 이름은 다음과 같다.

```
> names(iris)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

데이터 프레임 iris에서

- “pe”가 이름에 있는 변수 선택과
- “pe”가 이름에 있는 변수 제거를 실행해 보자.

우선 이름에 “pe”가 있는 변수를 함수 contains()를 이용해서 선택해 보면 마지막 변수인 Species 외에 대문자 “Pe”가 있는 변수도 함께 선택된 것을 볼 수 있다.

```
> select(iris, contains("pe"))
# A tibble: 150 x 3
  Petal.Length Petal.Width  Species
         <dbl>       <dbl>   <fct>
1         1.40       0.200   setosa
2         1.40       0.200   setosa
3         1.30       0.200   setosa
# ... with 147 more rows
```

이것은 함수 starts_with()와 ends_with(), 그리고 contains()에 공통적으로 적용되는 옵션 ignore.case의 영향인데 디폴트는 TRUE 값을 갖고 있어서 영문자의 소문자와 대문자를 구분하지 않는다. 만일 소문자 “pe”가 이름에 있는 변수를 선택하고자 한다면 다음과 같이 옵션에 FALSE 값을 주어야 한다.

```
> select(iris, contains("pe", ignore.case=FALSE))
# A tibble: 150 x 1
  Species
  <fct>
```

```
1 setosa
2 setosa
3 setosa
# ... with 147 more rows
```

이름에 “pe”가 있는 변수의 제거는 함수 contains() 앞에 마이너스를 붙이면 된다.

```
> select(iris,-contains("pe", ignore.case=FALSE))
# A tibble: 150 x 4
  Sepal.Length Sepal.Width Petal.Length Petal.Width
         <dbl>       <dbl>        <dbl>       <dbl>
1          5.1         3.5          1.4         0.2
2          4.9         3            1.4         0.2
3          4.7         3.2          1.3         0.2
# ... with 147 more rows
```

데이터 프레임을 구성하고 있는 변수들 중에 몇몇 변수를 제일 앞으로 옮겨서 다시 배치하고자 한다면 함수 everything()을 사용하면 된다. 예를 들어 데이터 프레임 iris에서 마지막 변수 Species를 첫 번째 변수로 자리를 옮겨 보자.

```
> select(iris, Species, everything())
# A tibble: 150 x 5
  Species Sepal.Length Sepal.Width Petal.Length Petal.Width
  <fct>          <dbl>       <dbl>        <dbl>       <dbl>
1 setosa          5.10        3.50         1.40       0.200
2 setosa          4.90        3.00         1.40       0.200
3 setosa          4.70        3.20         1.30       0.200
# ... with 147 more rows
```

- 변수 이름 수정: 함수 rename()

데이터 프레임의 변수 이름을 수정해야 하는 경우가 많이 있다. 이러한 경우 함수 select()를 사용할 수도 있으나, 함수 내에서 이름이 명시되지 않은 변수는 자동적으로 제거되기 때문에 불편한 면이 있다. 함수 select()를 사용하여 데이터 프레임 mtcars_t에서 변수 rowname을 Model로 이름을 바꿔 보자. 명시되지 않은 다른 변수는 모두 제거되었음을 볼 수 있다.

```
> select(mtcars_t, Model=rowname)
# A tibble: 32 x 1
  Model
  <chr>
1 Mazda RX4
2 Mazda RX4 Wag
3 Datsun 710
4 Hornet 4 Drive
5 Hornet Sportabout
# ... with 27 more rows
```

같은 작업을 함수 rename()으로 실행해 보자. 다른 변수가 모두 그대로 유지되어 있음을 알 수 있다.

```
> rename(mtcars_t, Model=rowname)
# A tibble: 32 x 12
  Model        mpg   cyl   disp    hp   drat    wt    qsec    vs    am   gear
  <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4   21.0  6.00   160  110    3.90   2.62  16.5   0     1.00  4.00
2 Mazda RX4~  21.0  6.00   160  110    3.90   2.88  17.0   0     1.00  4.00
3 Datsun 710  22.8  4.00   108   93.0  3.85   2.32  18.6  1.00  1.00  4.00
4 Hornet 4 ~  21.4  6.00   258  110    3.08   3.22  19.4  1.00  0     3.00
5 Hornet Sp~  18.7  8.00   360  175    3.15   3.44  17.0   0     0     3.00
# ... with 27 more rows, and 1 more variable: carb <dbl>
```

함수 select()와는 다르게 함수 rename()으로 이름을 수정하더라도 변수들의 순서는 그대로 유지된다.

```
> rename(mtcars_t, MPG=mpg, Model=rowname)
# A tibble: 32 x 12
  Model        MPG   cyl  disp    hp   drat    wt   qsec    vs    am   gear
  <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4   21.0  6.00   160  110    3.90   2.62  16.5   0     1.00   4.00
2 Mazda RX4~  21.0  6.00   160  110    3.90   2.88  17.0   0     1.00   4.00
3 Datsun 710  22.8  4.00   108   93.0  3.85   2.32  18.6   1.00  1.00  4.00
# ... with 29 more rows, and 1 more variable: carb <dbl>
```

7.4 새로운 변수의 추가: mutate()

데이터 프레임을 구성하고 있는 기존의 변수들을 이용하여 새로운 변수를 만들어 데이터 세트에 추가해야 할 경우가 종종 있다. 이러한 작업을 수행하는 함수가 mutate()이다. 함수에는 데이터 프레임이 첫 번째 입력 요소가 되고, 이어서 새로운 변수를 만드는 표현식이 차례로 입력된다. 새롭게 만들어진 변수는 데이터 프레임의 마지막 변수로 추가가 된다.

- 예제: mtcars

다음의 조건에 의하여 새로운 변수 kml과 gp_kml을 만들고 데이터 프레임에 첫 번째와 두 번째 변수로 추가해 보자.

  - 변수 kml: 1 mpg(mile per gallon)는 0.43 kml(kilometer per liter)
  - 변수 gp_kml: kml이 10 이상이면 'good', 10 미만이면 'bad'

변수 gp_kml을 만들 때 사용할 수 있는 함수로 base 패키지의 함수 ifelse()가 있다. 특정 조건의 만족 여부에 따라 두 가지 값 중 하나를 할당할 때 사용되며, 패키지 dplyr에는 이에 대응

되는 함수로 if_else()가 있다. 기본적인 사용법은 동일하지만 함수 if_else()의 경우에는 조건의 만족 여부에 따라 할당되는 두 가지 값의 유형이 같아야 한다는 제약 조건이 있으며, 그 결과로 생성되는 변수의 유형이 그대로 유지된다는 차이가 있다.

```
> mtcars_t <- as_tibble(mtcars)
> mtcars_t <- mutate(mtcars_t,
                     kml=mpg*0.43,
                     gp_kml=if_else(kml>=10,"good","bad"))
> select(mtcars_t, kml, gp_kml, everything())

# A tibble: 32 x 13
     kml gp_kml   mpg   cyl    disp    hp  drat    wt   qsec    vs    am
   <dbl> <chr>  <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>
1  9.03  bad     21.0  6.00   160   110    3.90  2.62  16.5  0     1.00
2  9.03  bad     21.0  6.00   160   110    3.90  2.88  17.0  0     1.00
3  9.80  bad     22.8  4.00   108    93.0  3.85  2.32  18.6  1.00  1.00
4  9.20  bad     21.4  6.00   258   110    3.08  3.22  19.4  1.00  0
5  8.04  bad     18.7  8.00   360   175    3.15  3.44  17.0  0     0
# ... with 27 more rows, and 2 more variables: gear <dbl>, carb <dbl>
```

만일 새로운 변수만 유지하고 나머지 변수는 모두 삭제하려고 한다면, 함수 transmute()를 사용하면 된다.

```
> transmute(mtcars_t,
            kml=mpg*0.43,
            gp_kml=if_else(kml>=10,"good","bad"))

# A tibble: 32 x 2
    kml  gp_kml
  <dbl> <chr>
1  9.03  bad
2  9.03  bad
3  9.80  bad
4  9.20  bad
5  8.04  bad
# ... with 27 more rows
```

7.5 그룹의 생성 및 그룹별 자료 요약: group_by()와 summarize()

변수들의 요약 통계량을 계산할 때 유용하게 사용되는 함수가 summarize()이다. 함수의 첫 번째 입력 요소는 데이터 프레임이고, 이어서 'name=fun'의 형태로 원하는 요약 통계량 값을 계산할 함수와 이 값에 부여할 이름을 입력하면 된다. 계산된 통계량 값에 이름을 부여해야 하는 이유는 함수 summarize()의 결과물이 tibble이어서 변수 이름이 필요하기 때문이다. 예를 들어 데이터 프레임 mpg의 변수 hwy의 평균값을 계산해 보자.

```
> summarize(mpg, hwy_mpg=mean(hwy))
# A tibble: 1 x 1
```

```
  hwy_mpg
    <dbl>
1    23.4
```

함수 summarize()에서 요약 통계량 값을 계산하는데 사용할 수 있는 함수는 반드시 결과가 숫자 하나로 출력되는 함수이어야 한다. 따라서 range()와 같이 길이가 2인 벡터를 결과값으로 출력하는 함수는 사용할 수 없고, mean(), sd(), min(), max() 등이 사용 가능한 함수이다. 또 다른 유용한 함수로써 만일 케이스의 개수를 세고자 한다면 함수 n()을 사용하면 되고, 함수 n_distinct()는 서로 다른 숫자의 개수를 세는데 사용할 수 있다.

```
> summarize(mpg, n=n(), n_hwy=n_distinct(hwy),
            avg_hwy=mean(hwy), sd_hwy=sd(hwy))
# A tibble: 1 x 4
      n n_hwy avg_hwy sd_hwy
  <int> <int>   <dbl>  <dbl>
1   234    27    23.4   5.95
```

함수 summarize()는 사실 그 자체만으로는 그렇게 특별히 유용한 함수라고 하기 어렵다. 그러나 전체 데이터를 몇 개의 그룹으로 구분하는 함수 group_by()와 함께 사용하면 매우 강력한 분석 도구가 된다.

함수 group_by()는 한 개 이상의 변수를 이용하여 전체 데이터를 그룹으로 구분하는 기능을 갖고 있다. 함수의 첫 번째 입력요소는 데이터 프레임이고, 이어서 그룹을 구분하는데 사용될 변수를 나열하면 된다. 실행 결과는 tibble이며, 실행 전의 tibble과 출력된 형태에서는 큰 차이를 볼 수 없으나 grouped_df라는 class 속성이 추가된 tibble이 된다.

예를 들어 데이터 프레임 mpg를 변수 cyl의 값으로 구분하고, 각 그룹별로 케이스의 개수와 변수 hwy의 평균값을 계산해 보자. 계산 절차는 우선 함수 group_by()로 mpg를 그룹으로 구분한 tibble을 만들고, 그것을 함수 summaize()에 통계량 계산을 위한 함수와 함께 입력하면 된다.

```
> by_cyl <- group_by(mpg, cyl)
> summarize(by_cyl, n=n(), avg_mpg=mean(hwy))
# A tibble: 4 x 3
    cyl     n avg_mpg
  <int> <int>   <dbl>
1     4    81    28.8
2     5     4    28.8
3     6    79    22.8
4     8    70    17.6
```

위 예제에서 함수 group_by()로 생성된 데이터 프레임 by_cyl은 분석의 중간 단계에 해당하는 데이터 프레임으로써 함수 summarize()에 입력되는 역할만을 담당하고 있다. 복잡한 분석을

진행하다 보면 이와 같이 중간 단계로 생성되는 객체들이 무수히 많을 수 있는데, 이것들에 이름을 붙이고 저장하는 것이 상당히 번거롭고 귀찮은 일이 된다. 다음 절에서 살펴보게 될 pipe 기능은 이러한 문제를 일소할 수 있는 매우 효과적인 것으로 앞으로 많이 사용하게 될 기능이다.

7.6 Pipe 기능

Pipe란 한 명령문의 결과물을 바로 다음 명령문의 입력요소로 직접 사용할 수 있도록 명령문들을 서로 연결하는 기능을 의미한다. 이것이 가능하다면, 분석 중간 단계로 생성되는 무수한 객체들을 따로 저장할 필요가 없기 때문에 프로그램 자체가 매우 깔끔해지며, 분석 흐름을 쉽게 이해할 수 있게 된다. Pipe 연산자 '%>%'는 tidyverse에 속한 패키지인 magrittr에서 정의된 것으로 tidyverse에 속한 다른 패키지에서도 사용이 가능하다.

기본적인 형태는 lhs %>% rhs로, lhs는 객체가 되고, rhs는 lhs를 입력요소로 하는 함수가 된다. 예를 들어 x %>% f는 객체 x를 함수 f()의 입력요소로 하는 f(x)를 의미한다. 만일 rhs에 다른 요소가 있다면 lhs는 rhs의 첫 번째 입력요소가 된다. 따라서 x %>% f(y)는 f(x,y)를 의미한다. 만일 lhs가 rhs의 첫 번째 요소가 아닌 경우에는 원하는 위치에 마침표(.)를 찍어야 한다. 예를 들어, x %>% f(y, .)은 f(y,x)를 의미한다.

이제 앞 절에서 살펴본 예제인 데이터 프레임 mpg를 변수 cyl의 값으로 구분하고, 각 그룹별로 케이스의 개수와 변수 hwy의 평균값 계산을 pipe를 이용하여 실행해 보자.

```
> mpg %>%
    group_by(cyl) %>%
    summarize(n=n(), avg_mpg=mean(hwy))
# A tibble: 4 x 3
    cyl     n  avg_mpg
  <int> <int>   <dbl>
1     4    81    28.8
2     5     4    28.8
3     6    79    22.8
4     8    70    17.6
```

- 예제: 데이터 프레임 airquality를 대상으로 다음의 문제를 해결해 보자.

  1) 월별로 변수 Ozone의 평균값

  2) 월별로 날수와 변수 Ozone에 결측값이 있는 날수, 그리고 Ozone이 실제 관측되어 결측값이 아닌 날수

  3) 월별로 첫날과 마지막 날의 변수 Ozone의 값

  4) 월별로 변수 Ozone의 가장 작은 값과 가장 큰 값

5) 월별로 변수 Ozone의 개별 값이 전체 기간 동안의 평균값보다 작은 날수

다섯 문제 모두 월별로 구분된 데이터 프레임이 필요하기 때문에 변수 Month로 그룹이 나뉘어진 데이터 프레임을 먼저 만들어 보자.

```
> airs_Month <- airquality %>%
   group_by(Month)
```

변수 Ozone에는 많은 수의 결측값이 있다. 따라서 함수 mean()에는 반드시 옵션 na.rm=TRUE를 함께 입력해야 결측값을 제외하고 평균값이 계산된다.

```
> # 1)
> airs_Month %>%
    summarize(avg_Oz=mean(Ozone, na.rm=TRUE))
# A tibble: 5 x 2
  Month avg_Oz
  <int>  <dbl>
1     5   23.6
2     6   29.4
3     7   59.1
4     8   60.0
5     9   31.4
```

월별로 날수는 함수 n()으로 알 수 있고, 결측값이 있는 날수는 함수 is.na()로 생성된 논리형 벡터를 함수 sum()에 입력하여 TRUE의 개수를 세면 알 수 있다. 또한 실제 관측이 있던 날수는 결측값이 아닌 날수이므로 함수 is.na()의 결과를 반대로 바꿔주고 함수 sum()에 입력하면 된다.

```
> # 2)
> airs_Month %>%
    summarize(n_total=n(), n_miss=sum(is.na(Ozone)),
              n_obs=sum(!is.na(Ozone)))
# A tibble: 5 x 4
  Month n_total n_miss n_obs
  <int>   <int>  <int> <int>
1     5      31      5    26
2     6      30     21     9
3     7      31      5    26
4     8      31      5    26
5     9      30      1    29
```

월별로 첫날과 마지막날의 변수 Ozone 값을 출력하기 위해서는 함수 first()와 last()를 이용해야 한다. 벡터 x에 대해서 first(x)와 last(x)는 각각 x[1]과 x[length(x)]의 결과가 출력되는데, 실제 두 함수는 대괄호에 의한 인덱싱보다 부가적인 기능이 더 있는 함수이다. 예를 들어 벡터 y를 두 번째 요소로 first(x, order_by=y)와 같이 입력하면 벡터 y로 결정된 순서에 의하여 해당 위치에 있는 벡터 x의 값을 선택한다.

```
> x <- c(2,4,6,8,10)
> first(x)
[1] 2
> first(x, order_by=order(-x))
[1] 10
```

유사한 기능을 가진 함수로 nth()가 있다. 벡터 x에 대해서 nth(x,2)는 x[2]의 결과가 출력되며, nth(x,-2)는 끝에서 두 번째 자료가 출력된다. 또한 예를 들어 길이가 3인 벡터에서 4번째 자료를 찾는 것처럼, 찾는 위치에 해당되는 자료가 없는 경우에는 NA가 아닌 다른 디폴트 값을 부여할 수 있다.

```
> x <- c(2,4,6,8,10)
> nth(x,2)
[1] 4
> nth(x,-2)
[1] 8
> nth(x,6)
[1] NA
> nth(x,6,default=99)
[1] 99
```

이제 월별로 첫날과 마지막날의 변수 Ozone 값을 출력해 보자.

```
> # 3)
> airs_Month %>%
    summarise(first_Oz=first(Ozone), last_Oz=last(Ozone))
# A tibble: 5 x 3
  Month first_Oz last_Oz
  <int>    <int>   <int>
1     5       41      37
2     6       NA      NA
3     7      135      59
4     8       39      85
5     9       96      20
```

월별로 변수 Ozone의 max와 min 값을 출력해 보자. 두 함수도 옵션 na.rm=TRUE을 입력해야 결측값을 제외하고 최대와 최소를 계산할 수 있다.

```
> # 4)
> airs_Month %>%
    summarize(max_Oz=max(Ozone, na.rm=TRUE),
              min_Oz=min(Ozone, na.rm=TRUE))
# A tibble: 5 x 3
  Month max_Oz min_Oz
  <int>  <dbl>  <dbl>
1     5  115     1.00
2     6   71.0  12.0
3     7  135     7.00
4     8  168     9.00
5     9   96.0   7.00
```

마지막 문제로 월별로 변수 Ozone의 개별 값이 전체 기간 동안의 평균값보다 작은 날수를 세어 보자.

```
> # 5)
> m_Oz <- with(airquality, mean(Ozone, na.rm=TRUE))

> airs_Month %>%
    summarize(low_Oz=sum(Ozone<m_Oz, na.rm=TRUE))
# A tibble: 5 x 2
  Month low_Oz
  <int>  <int>
1     5     24
2     6      8
3     7      8
4     8     10
5     9     22
```

7.7 7장을 마치며

데이터 분석 과정에서 데이터 다듬기의 중요성은 우리 모두 잘 알고 있는 사항이다. 그러나 1부에서 소개되었던 전통적인 방식은 일관된 법칙이 없어서 사용하는 함수마다 사용법을 따로 익혀야 하는 문제가 있었다. 이에 비하여 패키지 dplyr의 데이터 다듬기 함수들은 모두 공통적인 구조를 가지고 있고 사용이 매우 편하여, 우리가 원하는 결과를 어렵지 않게 만들 수 있게 되었다.

이제 다음 장에서는 패키지 tidyverse 중에서 가장 많이 사용되는 패키지인 ggplot2에 의한 데이터 시각화를 살펴보고자 한다. R이 얼마나 강력한지를 다시 한번 느낄 수 있는 기회가 될 것이다.

7.8 연습문제

1. 3장 10번 연습문제를 패키지 dplyr의 함수를 이용하여 해결해 보자. 데이터 프레임 airquality에는 6개 변수, 153개 케이스의 데이터가 있으며, 많은 수의 결측값도 포함되어 있다.

   1) 변수 Wind의 값이 mean(Wind) 이상이고 Temp가 mean(Temp) 미만이 되는 케이스만을 선택하여 air_sub1에 할당하라. 변수는 Ozone과 Solar.R, 두 변수만을 선택한다.
   2) 변수 Wind의 값이 mean(Wind) 미만이고 Temp가 mean(Temp) 이상이 되는 케이스만을 선택하여 air_sub2에 할당하라. 변수는 Ozone과 Solar.R, 두 변수만을 선택한다.

3) 두 데이터 프레임 air_sub1과 air_sub2에 있는 두 변수 Ozone과 Solar.R의 평균값 및 케이스의 개수를 계산하여 다음과 같은 결과를 얻도록 하라.

air_sub1의 경우			air_sub2의 경우		
	n	m_oz	m_solar	n	m_oz	m_solar
1	42	17.61765	165.85	55	71.36364	204.0385

2. 3장 11번 연습문제를 패키지 dplyr의 함수를 이용하여 해결해 보자. 데이터 프레임 mtcars는 1974년 발행된 어떤 자동차 잡지에 실린 32대 자동차의 성능에 관한 자료로서 11개 변수로 이루어져 있다.

1) mtcars의 row name을 변수 model로 전환하여 변수로 추가하고, 처음 3 케이스를 다음과 같이 출력하라.

```
# A tibble: 32 x 12
  model      mpg   cyl  disp    hp    drat    wt  qsec    vs    am
  <chr>    <db1> <db1> <db1> <db1> <db1> <db1> <db1> <db1> <db1>
1 Mazda R~  21      6   160   110    3.9   2.62  16.5    0     1
2 Mazda R~  21      6   160   110    3.9   2.88  17.0    0     1
3 Datsun ~  22.8    4   108    93   3.85   2.32  18.6    1     1
# ... with 29 more rows, and 2 more variables: gear <db1>,
#   carb <db1>
```

2) 변수 model, mpg, cyl, disp, hp, wt, am만을 선택하여 데이터 프레임 cars를 만들어라.

3) 변수 disp는 세제곱인치 단위의 배기량이다. 이것을 cc단위의 배기량으로 변환한 변수 disp_cc를 만들어 데이터 프레임 cars에 추가하고, 변수 disp는 제거하라. 단, 1 세제곱인치=16.4 cc.

4) 변수 disp_cc를 이용하여 아래의 규칙으로 변수 type을 생성하여 cars에 추가하고, 처음 3 케이스를 출력하라.

disp_cc <1000	type="Compact"
1000 ≤ disp_cc < 1500	type="Small"
1500 ≤ disp_cc < 2000	type="Midsize"
disp_cc ≥ 2000	type="Large"

5) 변수 am이 1이고 cyl이 8인 자동차들의 mpg, disp_cc, type의 값을 다음과 같이 출력하라.

```
# A tibble: 2 x 4
  model            mpg disp_cc  type
  <chr>          <dbl>   <dbl> <fct>
1 Ford Pantera L  15.8   5756. Large
2 Maserati Bora   15     4936. Large
```

6) 변수 cyl의 값에 따라 구분되는 자동차 대수 및 mpg, disp_cc, hp, wt의 평균값을 다음과 같이 출력하라.

```
# A tibble: 3 x 6
    cyl     n   mpg  disp_cc    hp    wt
  <dbl> <int> <dbl>   <dbl> <dbl> <dbl>
1     4    11  26.7   1724.   82.6  2.29
2     6     7  19.7   3006.  122.   3.12
3     8    14  15.1   5791.  209.   4.00
```